Title:Multiprocessor computer system having multiple coherency regions and software process migration between coherency regions without cache purges.

FIELD OF THE INVENTION: This invention relates to a multiprocessor computer system having a plurality of nodes and dynamic cache coherency regions. This invention relates particularly to such a computer system that can move software processes between coherency regions without requiring a selective purging of cache contents.

Related Applications:

This invention is related to an application entitled: Multiprocessor System with Dynamic Cache Coherency Regions, USSN _____, filed _____, filed concurrently herewith and referenced herein.

This co-pending application and the present application are owned by one and the same assignee, International Business Machines Corporation of Armonk, New York.

The description set forth in these co-pending application is hereby incorporated into the present application by this reference.

Trademarks: IBM ® is a registered trademark of International Business Machines Corporation, Armonk, New York, U.S.A.. Other names may be registered trademarks or product names of International Business Machines Corporation or other companies.

Background of the invention:

The idle time spent by computer processors while waiting for memory references to complete has become a much larger fraction of the total execution time for a wide variety of important commercial and technical computing workloads. Many prior-art techniques have been used in multiprocessor system designs to minimize the time a processor must wait while the access of main storage locations is completed. These techniques fall broadly into two categories. The first category of techniques attempts to find additional instructions for the processors to execute while waiting for the memory reference which is experiencing a delay. These techniques include such hardware and software mechanisms as out of order execution and multithreading. The second category of techniques focuses on minimizing the latency of the memory reference itself, e.g. SRAM caches, DRAM caches and high speed multiprocessor bus architectures. SRAM and DRAM caches have been extremely successful in reducing memory reference latency and one or both are used by all current multiprocessor designs. Prior-art cache designs include specialized hardware and software which maintain cache coherence for multiprocessor systems. For systems which connect a plurality of processors via a shared bus, a snoop bus protocol is typically employed. Each coherent transaction performed upon the shared bus is examined (or "snooped") against data in the caches of all other devices attached to the bus. If a copy of the affected data is found, the state of the cache line containing the data may be updated in response to the coherent transaction.

Although caches have worked well for multiprocessor systems with a moderate number of processors, prior-art multiprocessor designs do not scale well when extended to large numbers of processors for many important workloads including the transaction and database workload simulated by the TPC-C benchmark.

Logical partitioning, as described in US Patent No. 4843541, when using shared processors also causes poor scaling for

prior-art system designs when extended to large numbers of processors. US Patent No. 4843541 shows how a virtual machine hypervisor program can be used to "partition the resources in a central electronic complex of a data processing system into a plurality of logical partitions". Logical partitioning is widely used on large multiprocessor systems to run many workloads that operate on private data simultaneously. In a typical system employing logical partitioning, an operating system instance is initialized within each logical partition. The logical partition can have from 1 to n logical processors. The hypervisor is responsible to dispatch each of the logical processors onto a physical processor. If a physical processor is the host of just a single logical processor over a long period of time it is said to be "dedicated" to that logical processor's partition. If a physical processor is the host of the logical processors from multiple partitions it is said to be a "shared" processor. It is desirable, from an overall hardware utilization point of view, for a large multiprocessor system to allow the flexibility of defining many or most of the physical processors as "shared" and allowing the movement of logical processors among the physical processors of the multiprocessor as the utilization of the physical processors fluctuates with external changes. Prior-art multiprocessor cache designs do not scale well for these partitioned workloads, especially when the physical processors are defined as "shared".

A large factor in the poor performance scaling of large multiprocessors for both the large single database workload and the shared logical partition case is the relationship between increasing numbers of processors and the time delay required to communicate among them. Snoop bus protocols require memory references that miss local caches to be broadcast to all caches which may contain a copy of the requested lines, typically all

other caches in the system. The bus bandwidth required to distribute the addresses and responses for large multiprocessor systems is very high. The need to provide the required high bandwidth has driven prior-art designs to use switch chips with many wide ports, expensive chip carriers to provide the needed pins, expensive card technology to provide good electrical characteristics and therefore high speed buses, expensive card connectors to provide wide buses etc. The cost of all these elements has become a significant problem when trying to improve the cost/performance of large multiprocessor systems.

Prior-art designs have attempted to solve these two problems, coherency operation latency and address bandwidth limitations, in many different ways but each has imposed other costs on the system design which the current invention seeks to avoid.

Large shared caches, as exemplified in the IBM S/390 G4 design (IBM Journal of Research and Development Volume 41, Numbers 4&5, 1997) have been used in prior-art designs to address both problems. The interconnection of a few large shared caches does provide good latency for requests which hit in the shared cache. The inclusive shared cache also acts as a filter which eliminates the need to broadcast addresses to all of the processors in the system for some cases. The design does not scale well to large numbers of processors. The use of additional processors drives the design to using large multichip modules with many wiring layers and L2 cache chips with an extremely large number of I/O required to provide a port for each of the processors connected.

Multiprocessor systems which rely on directories to track the access of local memory by remote requesters, as exemplified by the Sequent NUMA-Q design ("STiNG: A CC-NUMA Computer System for the Commercial Marketplace", in Proc. 23rd International Symposium of Computer Architecture, May 1996) work to reduce the

address bandwidth required for large numbers of processors. They do so at the expense of large RAM directories and an increase in protocol complexity and hardware support. This type of design also depends upon an assumption that the majority of the main storage lines referenced by a particular software process is located on the same physical node as the node that the processor that is executing the workload is currently dispatched on. There are severe performance penalties for cases where a workload is accessing a large number of remote lines since the number of lines that can be "checked out" by remote nodes is limited by the size of the NUMA directories. One goal of the current invention is to allow the movement of the execution of a workload quickly and easily among many processors without the need to move main storage contents and without significant performance degradation.

Hagersten et al., US Patent No. 5852716 describes the use of multiple address partitions in order to define cache coherent operations which are either "local" and confined to a subset of processors in a large multiprocessor or "global" and therefore broadcast to all processors. A local transaction in Hagersten is defined as one which has physical memory allocated to the same subset of processing nodes as the subset to which the processor which originates the storage request belongs. The description beginning on in 63 of column 7 of US Patent No. 5852716 makes it clear that this prior-art invention does not allow the movement of a process between what is referred to as "local domains" without either moving the physical storage associated with that process or by changing the addressing mode to "global".

We have determined that there is a need for techniques to reduce transmission of address requests between various processors in a multiprocessor computer system without using large amounts of SRAM directory and without requiring the movement of main storage contents. In developing solutions for fulfilling this need we have determined that there is an

associated need to reduce the latency of all storage reference transactions in large multiprocessor systems.

Summary of the Invention:

In fulfilling these determined needs, we have hardware coherency controls which enable a system which uses multiple cache coherency regions to operate without the use of cache purges during some operations which move software processes between coherency regions. The current invention works for the case where a software process is moved out of one coherency region that is no longer going to be used and into another that has been created to cover the same address space as the first but which will include a new set of processing nodes. The preferred embodiment of our invention works to allow a supervisor program to move a software process from one coherency region encompassing one set of processing nodes to another coherency region encompassing another set of processing nodes without requiring cache purges of caches in any of the processing nodes. If the destination coherency region contains fewer hardware processing nodes than the original coherency region, the size of the coherency region has been effectively been reduced.

The preferred embodiment of the invention is embodied in a multiprocessor computer system having a plurality of nodes and which uses a table of active coherency region information associated with each processing node to determine when to alter the prior-art cache state transitions. A supervisor program initializes the tables associated with each processing node. An entry in the table is made for each coherency region that the supervisor program intends to use on that processing node. Each coherency region is assigned a unique coherency region ID which the supervisor can associate with all the software processes that

access storage addresses that are encompassed by the coherency region.

Processing nodes which use the invention are able to identify incoming storage requests which target lines that are no longer part of the address space of any software process that is currently enabled by the supervisor software to be dispatched on the node. In the preferred embodiment this information allows a processing node to identify cache lines that are no longer actively used by any software processes on that node and to change the cache entries to invalid in response to a storage request from outside the coherency region.

The advantages of the invention are numerous. One advantage of the invention is that it eliminates the need for cache control hardware that would otherwise be required to perform selective purges of caches when moving software processes between cache coherency regions. A second advantage is that the invention allows all of the caches in a system to continue processing coherency transactions while the coherency boundaries for a software process are effectively changed. A third advantage is that cache lines belonging to a software process that is no longer actively being dispatched on a given node can be identified and invalidated thereby enabling their reuse.

These and other improvements are set forth in the following detailed description. For a better understanding of the invention with advantages and features, refer to the description and to the drawings.

Description of the Drawings:

FIGURE 1 illustrates a block diagram of a computer with dynamic coherency boundaries. The node controller contains a hardware implementation of the Active Coherency Region Table.

FIGURE 2 shows how multiple instances of node of the computer from Figure 1 can be connected with a second level controller to create a large multiprocessor system.

FIGURE 3 shows a single processing element from Figure 1.

FIGURE 4 illustrates a table that describes how the node controller uses the mode bits to determine which processors must receive any given transaction that is received by the node controller.

FIGURE 5 shows a table that describes how the second level controller uses the mode bits to determine which nodes must receive any given transaction that is received by the second level controller.

FIGURE 6 shows one possible mapping of logical partitions to allowable physical processors.

FIGURE 7 shows additional detail of a hardware implementation of the Active Coherency Region Table. The table is a supervisor software controlled list of the coherency regions which are currently allowed to use the processing node for software process dispatch.

Our detailed description explains the preferred embodiments of our invention, together with advantages and features, by way of example with reference to the drawings.

Detailed Description of the Invention:

Turning now to Figure 1, a block diagram of one embodiment of one node (10) of a computer with dynamic coherency boundaries is shown. Figure 1 shows a plurality of processors P0-P3, each with a cache, attached to a local node controller (11). The local controller connects multiple processors together with a DRAM main storage element (12). Storage transactions that are initiated by

a single processor are transmitted to the node controller which may in turn transmit the transaction to any or all of the other processors in the node. The node controller may also transmit the transaction on bus (13) to other parts of the computing system which contains additional processors (not shown). The Active Coherency Region Table (14) is used by the node controller to determine the proper cache state transitions required in response to storage requests which are entering the node on the bus (13) from other parts of the computing system (not shown). Figure 2 shows how multiple instances of node (10) from Figure 1 can be connected with a second level controller (15) to create a large multiprocessor system. Figure 1 shows the use of 4 processing elements but it should be understood that any number of processing elements could be used. Figure 1 shows only 1 memory element but it should be understood that any number of memory elements could be used. The preferred embodiment uses the hierarchical bus organization shown in Figures 1 and 2, but the invention can be applied to multiprocessor systems that use any other type of interconnect topology.

Figure 3 shows a single processing element from Figure 1. The invention uses one or more coherency mode bits (16) for each processor in the multiprocessor system. The invention uses a coherency region ID for each processor in the multiprocessor system. The coherency mode bits and coherency region ID associated with a processor are sent together with each storage transaction that is initiated by that processor when the transaction is transmitted to the node controller via bus (17) on figure 3. It should be understood that a node controller is used in this embodiment but could be replaced with a simple physical bus in other embodiments. The cache coherency hardware in node controller (11) and second level controller (15) use the mode bits associated with each transaction to determine which caches must participate in any storage transactions that they receive

from any of the processors. The preferred embodiment uses 3 mode bits. The 3 mode bits are used together to identify the following modes of operation for the node controller and secondary controller. A coherency mode setting of "000" is used to define a coherency region of just a single processor as shown by dashed lines (10') in figure 1. Any of the other 3 processors could be used in a single processor coherency region also. A coherency mode setting of "001" is used to define a coherency region of two processors as shown by dashed lines (18) and (19) in figure 1. The current embodiment allows the hypervisor to define two-processor coherency regions that cover either (P0 and P1) or (P2 and P3) in order to simplify the hardware controls required in the node controller. Other embodiments could allow other combinations, such as P0 from node 1 and P0 from node 2. A coherency mode setting of "010" is used to define a coherency region that includes all of the processors of a single node as shown by dashed line (20) in figure 1. A setting of "101" defines a coherency region that includes two nodes as shown by dashed lines (21) and (22) in figure 2. Finally, a processor with a setting of "111" indicates that all storage transactions generated must be sent to all the caches in the entire system.

The coherency mode setting is considered part of the state of a logical partition and therefore part of the state of the logical processors which are defined in that partition. In the current embodiment, all logical processors from a single logical partition have the same coherency mode setting at a single point in time. It should be understood that additional software or firmware could be used to define processes within a single partition which use an isolated set of storage addresses and therefore could be provided a different coherency mode setting and a different set of allowable physical processors to be used for dispatch. When a logical processor is dispatched onto a physical single processor the physical processor temporarily

takes on the coherency mode setting of the logical processor. The coherency mode bits are sent with all storage transactions generated by the processor when they are transmitted to the node controller (11). Since many logical partitions can be defined and used at once, many different and overlapping coherency regions are used at the same time. The current invention provides hardware and firmware controls in the node controller (11) and second level controller (15) which use the coherency mode bits that accompany each bus transaction to determine how to route the transaction over the buses which interconnect the processors in the system.

Figure 4 shows a table that describes how the node controller uses the mode bits to determine which processors must receive any given transaction that is received by the node controller. Figure 5 shows a table that describes how the second level controller uses the mode bits to determine which nodes must receive any given transaction that is received by the second level controller. Figure 6 shows one possible mapping of logical partitions to allowable physical processors. In the current embodiment, the node controller will forward all transactions received from the secondary node controller to all the processors connected to the node controller. It should be understood that there are many other potential coherency boundaries that could be established which would require the node controller to transmit requests which come from the second level node controller to just a subset of processors connected to the second level node controller.

The supervisor software creates a unique Coherency Region ID for each process that has its own coherency region. The supervisor software creates a table for each processing node in the system. The table has an entry for every Coherency Region ID

that is currently allowed to be dispatched on that processing node.

Alternative approaches may be found in the related patent application. The referenced related patent application uses a combination of software control programs and hardware mode bits to define dynamic coherency boundaries in a computing system which utilizes more than one processing node. The coherency boundaries can be adjusted to create coherency regions that cover any number of processing nodes, from one node to all the nodes in the entire system. The related application also describes how multiple coherency regions can be defined, each operating on a private address space. The coherency regions can be expanded to include additional nodes at any time during system operation. The coherency regions can also be reduced in size by removing a processing node from that region, while following the required procedures. Included in those procedures is the need to purge some cache entries in the processing node which is about to be removed from the coherency region. The cache entries which need to be removed are only those that hold cached copies of main storage lines that are part of the coherency region that is being reduced in size. Caches which are unable to perform a selective purge based upon the identification of the coherency region that "owns" the cached line must be purged completely. The selective purges require additional cache controller hardware as compared to prior-art designs. The preferred embodiment illustrated here is applicable to a processing system taking advantage of the ability to eliminate the need for the cache purges when moving a software process between two distinct sets of processing nodes.

The related patent application describes how the supervisor software must change the "cache coherence mode" of the processor when dispatching a software process that uses a cache coherency

region that is different than previously dispatched software process. The current invention requires that a storage request be tagged with more specific information about the precise coherency region that originated the request. The current invention uses the Coherency Region IDs of the originating process as this more specific tag. In the preferred embodiment the "cache coherency mode" is still used by the node control logic to determine which nodes in a multiprocessor system are required to examine a storage request. The coherency region ID of the current invention is used by the snooping processors to determine the correct snooping response.

It should be understood that an alternative preferred embodiment of the current invention could use the coherency region ID to perform the function of the "cache coherency mode" in addition to the function described for the preferred embodiment. The preferred embodiment makes the assumption that the node controller logic that must read the "cache coherency mode" can be made to be faster and smaller when it can rely on the simple encoding of the "cache coherency mode". The alternative preferred embodiment of this invention provides programmable logic which is found in the node controller which the supervisor program uses to help the node controller determine which physical nodes are associated with specific coherency region IDs.

The current invention alters the logic which is used to decide which processing nodes must examine any particular storage request, as compared to the related patent application. In the referenced related patent application the coherency region, as expressed by the mode bits, were used to make the determination. In the current preferred embodiment of the invention this is changed so that any storage request which misses all of the caches in the originator's coherency region is then sent on to

all processing nodes in the entire system, regardless of the setting of the mode bits. Requests which hit in the originator's coherency region but which do not have the correct cache state do not need to be sent outside the coherency region. An example of this latter case is a storage request which intends to alter a line of storage but which finds during the course of a storage transaction that a cache within its coherency region has a copy of the line that is marked as shared. The cache state transitions of the current invention are set up to ensure that a cache line cannot be marked as shared in two separate coherency regions. When the supervisor program is moving a coherency region from one set of processing nodes to another set of processing nodes it is effectively leaving behind cache entries for the coherency region on the old nodes. The current invention works to ensure that these old cache entries will be seen by requests originating from the new processing nodes and that cache entries for the same main storage addresses will not be established in the new processing nodes until the old entries are invalidated.

The current invention provides for the use of additional communication between processing nodes as compared to the system described in the related patent application. It is assumed that the current invention will be used in conjunction with large on-node caches that will minimize the number of storage requests which completely miss the on-node caches. The current invention will still enable an overall reduction in the communication between processing nodes as compared to a prior-art design which does not use dynamic coherency boundaries.

The current invention provides for easy movement of software processes between coherency regions. With the use of supervisor software the invention enables a cache controller in a processing node to be sure that no copy of a requested storage address exists outside that processor's current coherency region, as

specified by the current coherency region mode, whenever a cache entry for the requested storage address is found to exist on any cache in the processing node that contains the processor that initiated the request.

The new coherency region ID is used by the snooping caches to decide how to respond to a storage request. If the coherency region ID attached to an incoming request does not match any of the IDs in the processing node's active coherency region table then the caches in the processing node respond with the usual MESI protocol response and then set the cache to invalid. If the change to invalid requires dirty data to be written back to main storage then that operation is also started at the same time. Figure 7 shows an example of an Active Coherency Region ID table that is used by a processing node during incoming storage transaction processing. The table can be made arbitrarily large to accommodate the expected number of entries, one per unique coherency region that is active on this processing node. The preferred embodiment uses a hardware table but it should be clear that any combination of software, firmware or hardware could be used to implement the required function for the invention.